SNAPP: Stateless Network-Authenticated Path Pinning

Bryan Parno, Adrian Perrig, David Andersen

September 19, 2007
CMU-CyLab-07-013

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

SNAPP: Stateless Network-Authenticated Path Pinning

Bryan Parno, Adrian Perrig, David Andersen
Carnegie Mellon University

ABSTRACT

This paper examines a new building block for next-generation networks: SNAPP, or Stateless Network-Authenticated Path Pinning. SNAPP-enabled routers securely embed their routing decisions in the packet headers of a stream of traffic, effectively pinning a flow's path between sender and receiver. A sender can use the pinned path (even if routes subsequently change) by including the path embedding in later packet headers. This architectural building block decouples routing from forwarding, which greatly enhances the availability of a path in the face of routing misconfigurations or malicious attacks. To demonstrate the extreme flexibility of SNAPP, we show how it can support a wide range of applications, including sender-controlled paths, expensive route lookups, sender anonymity, and sender accountability. Our analysis shows that SNAPP's overhead is low, and the system is easily implemented in hardware. We believe that SNAPP is a worthy addition to the network architect's toolbox, enabling a variety of new designs and trade-offs.

1. INTRODUCTION

In decades past, debates abounded over the relative merits of circuit-switched vs. packet-switched networks, with each camp claiming unbeatable technical advantages. With the recent expansion of community interest in novel Internet architectures, the discussions are beginning again [14].

Recognizing the merits of both approaches, we propose a new technique that achieves the advantages of both, at the expense of increased packet state. As available bandwidth continues to increase dramatically, we believe this represents a beneficial tradeoff.

Specifically, we consider a *path pinning* approach. Like a circuit-switched network, path pinning binds a particular flow (or connection) to a specific path through the network for as long as the flow and the path both exist. Like a packet-switched network, the path pinning mechanism we present does not require per-flow state in the network. Prior techniques for providing path pinning rely on copious per-flow state in routers, or provide end hosts with an uncomfortable degree of control (see §2).

The contribution of this work is a *stateless*, *secure* path-pinning building block that provides numerous benefits without the drawbacks of prior approaches. This mechanism, SNAPP, adds a small amount of information to packet headers as they pass through the network, akin to the techniques used by network capabilities [15, 23, 25]. Once a packet has traversed a path, the sender and receiver can send additional packets that are forwarded based upon the information encoded in the packet. Of course, routers may always make a new routing decision; §4.5 shows how to seamlessly update the path pinning while implicitly informing the sender and receiver of the change.

We explore SNAPP's suitability as a basic building block for use at the network layer, as well as at higher-layers such as overlay networks. Our evaluation in §6 indicates that we can implement SNAPP using today's hardware, and that SNAPP is a useful tool for new network architectures.

SNAPP's versatility addresses many problems:

Expensive Route Lookups. Because SNAPP decouples route table lookups and forwarding, a designer might consider an architecture in which comparatively expensive routing lookups are amortized over subsequent packets. As an extreme example, the Intentional Naming System [1] performs complex string-matching and range comparisons when computing packet destinations. SNAPP provides an attractive optimization for data flow in such systems.

Decoupling Forwarding from Routing Failures. By placing the results of routing decisions into packet-carried state, SNAPP makes it possible to send subsequent packets without depending on per-hop routing calculations. This approach greatly improves the availability of forwarding in the face of routing misconfigurations or malicious attacks. Even if the routing infrastructure is temporarily broken, communicating parties can continue to use established connections.

Efficient Packet Forwarding. SNAPP forwarding requires only sequential processing of header information, without requiring complex routing table lookups or even high-latency memory lookups. Such streamlined packet processing enables the construction of high-speed forwarding nodes. The processing could even be fully performed by purely optical switching elements.

Sender-Controlled Paths. Systems such as NIRA [24] and Platypus [16] give senders more control over the route their packets take through the network. These systems typically use a mechanism such as source routing to provide senders with this control. SNAPP separates the forwarding plane from the mechanism by which paths are established, potentially allowing *all* of these solutions to co-exist on the same network.

Capability-Based Networks. A drawback of capability-based designs [15,23,25] is their fragility under path changes. Path pinning binds flows to their original paths, so communication is not interrupted as long as the physical path remains intact, thus enhancing the robustness of most existing capability systems.

Load Balancing. Fine-grained load balancing is complicated by protocols such as TCP that perform poorly when packets are re-ordered [26]. Load balancing techniques therefore often pin paths to ensure good TCP performance.

Sender Anonymity. In §5.1, we examine a bidirectional variant of SNAPP, which stores in packets both forward and reverse path information, to support sender anonymity.

Sender Accountability. Surprisingly, SNAPP can also be used to create the polar opposite: a system for assigning responsibility for every packet and its contents to the originating sender. While we do not necessarily support creating such a system (and it would require infrastructure in addition to SNAPP), we believe SNAPP's ability to support applications at two extremes of the design space demonstrates its flexibility and suitability as a basic building block.

2. RELATED WORK

2.1 Capability-based Architectures

SNAPP encodes path pinning information into packets in a manner inspired by network capabilities. While capabilities resemble our path token, the capabilities are used purely to authorize packet transmission, not to encode per-hop forwarding information.

As we discuss further in §7, SNAPP complements rate-limiting and DoS-preventing capability architectures by reducing the problems they have with route changes. Since both types of systems use cryptographically encoded information in packets, they would be ideal candidates to integrate with SNAPP.

2.2 Path Pinning via Virtual Circuits

One effect of a virtual circuit (VC) architecture is that connections are typically pinned to the path they traversed during setup, though in some systems they may be re-bound, particularly in the case of failure. VCs meet many of the requirements we express for a path pinning system in §3: They are unforgeable, because all of the intelligence about the path is maintained in the network. They permit the network complete control over the path that is used. They can be very efficient because of their fast lookups (e.g., MPLS [18]) and they may require less space in the packet header than an equivalent packet-switched network.

However, static virtual circuits must be explicitly preconfigured by provisioners, typically in response to heavy-weight, or even out-of-band, requests from customers, making the overhead prohibitive for brief, transient communication. With both static and dynamic virtual circuits, switches typically maintain per-circuit state, even with lighter-weight IP-based protocols such as RSVP [8]. Aggregation is possible with protocols like MPLS, but MPLS typically operates within a single domain.

2.3 Packet Switching and Path Pinning

Current IP forwarding tightly couples routing and forwarding. The router performs a route table lookup for every packet it receives and then forwards the packet out of the chosen interface. There is no provision for the router to include routing information in the packet. The need to perform a routing lookup (or calculation) for every packet limits the complexity of routing algorithms, since the algorithm should not constrain forwarding speed. By closely coupling routing and forwarding, traditional IP forwarding also couples their failure modes: routing failures inevitably result in forwarding failures. Moreover, the sender has no control over the path a packet traverses.

One oft-discussed alternative to destination-based routing is *source routing*, which can provide a pinned path. Unfortunately, traditional IP strict source routing has several shortcomings. Sources must understand the entire network topology to create a hop-by-hop path to the destination. Loose source routing mitigates this requirement, but sacrifices path pinning in the process. Source routing implicitly trusts sources to construct valid, allowed paths; routers have little or no control over whether such paths should be permitted.

Platypus [16] uses a capability-like mechanism to allow end systems to request routes from ISPs and to enable the routers to verify that the use of that route is permitted. In contrast, SNAPP's markings are generated on the path by the routers themselves, requiring no coordination between routers in an ISP. We believe that SNAPP would be an effective generic mechanism upon which to build a Platypus-like system. In fact, the developers of Platypus note that a "[way] for Platypus routers to cache forwarding directives for traffic flows" would be a valuable addition to their protocol. SNAPP provides exactly such a mechanism without requiring any state on the router.

A number of projects, such as Platypus, NIRA [24], and RON [2] have shown that many benefits arise from source-influenced path selection; we discuss some of these possibilities in more detail in §5.4. Like Platypus, WRAP [3] uses a form of loose source routing to specify a domain-

level path through the Internet. WRAP routers explicitly fill in the reverse path as the packet progresses through the network. Unlike Platypus and SNAPP, WRAP's header is not authenticated. Note that while SNAPP enables senders to select amongst available network paths, it prevents arbitrary source routing that would violate router policies.

Overlay networks are another popular approach to providing control over paths. Routing overlays such as RON [2], the X-Bone [22], and i^3 [19] permit end hosts limited control over forwarding by sending packets indirectly through other end-hosts. In contrast, SNAPP is intended to provide router-level path pinning in the forwarding path. Routing overlays could take advantage of path pinning in a number of ways (see the discussion in §5.4).

LIRA [20] binds packets to a particular path by computing a packet label based on the XOR of the IP addresses of the routers along the path. Since these labels are used within a single ISP, the path identifiers are not secured. LIRA also requires per-path router state.

3. DESIGN REQUIREMENTS

At a high level, a dynamic path-pinning system allows routers to insert information about their routing decisions into each packet they receive. Senders can then include this information in subsequent packets, allowing the routers to forward the packets without performing a routing calculation. Below, we outline the requirements for such a scheme.

Unforgeable Paths. End-hosts should not be able to construct arbitrary network paths, nor to recombine parts of established paths. While a malicious router can always misdirect a packet, we require a legitimate router to detect such a deviation and drop the packet. The first constraint prevents hosts from violating the routing policy at the routers; the second prevents subverted network elements from hijacking legitimate flows.

Local Router Operation. Each router should act using only local information. That is, the system should require no coordination between routers. In addition, the system must not require new trust relationships between an end host and the routers or between the routers themselves. These requirements ease deployment and improve security, since each router only trusts itself. Furthermore, a compromised router only affects its own traffic; it cannot influence another legitimate router.

Dynamic Configuration. The path pinning mechanism must not impose additional configuration effort nor require statically constructed routes. Excessive configuration discourages adoption and limits the number and types of routes to which the system would apply.

Controllable Topology Disclosure. The basic building block should be flexible in terms of the amount of information it reveals about the network topology. Some applications may wish to keep the topological information completely opaque, while others may wish to reveal it

Minimal Packet Overhead. To conserve bandwidth, the information inserted into the packets must not consume excessive space in the packets.

No Per-Flow Router State. Eliminating per-flow state at routers decreases the cost of routers, both because less high-speed memory is required and because it simplifies the router's design and implementation. It also improves the scheme's scalability.

Efficient Forwarding. A viable forwarding scheme must support efficient packet forwarding at current and future line rates. The high latency of memory accesses suggests that we minimize memory lookups. This approach will also enable all-optical packet forwarding, which will become important in the near future.

4. SNAPP

We begin with a high-level description of SNAPP. We then explain how senders create a pinned path and how routers use the information in a path pinning to forward packets. Then, we show in detail how to preserve the integrity of a pinned path. Finally, we explain how routers maintain and update the pinned paths. For clarity, our detailed description of the cryptographic mechanisms underlying SNAPP begins with a simplified version in §4.4, which we revise through the following subsection to add additional properties. We present the bit-level details of the fields in a SNAPP packet's header in §4.6.

4.1 Protocol Overview

Figure 1 illustrates the SNAPP protocol. A sender initiates path pinning by sending a SETUP packet to the receiver. Each router on the path makes its usual routing calculation and embeds the result in the setup packet, along with an authenticator. Together, the decision and the authenticator form a segment embedding. The receiver returns the accumulated set of segment embeddings, called a path embedding, to the sender. The sender includes the path embedding in subsequent packets.

When a router receives a packet containing a path embedding, it locates its own segment embedding, verifies the authenticator, and then forwards the packet based on the routing decision encoded in the segment embedding. Note that SNAPP data can be piggybacked on top of existing network traffic. For instance, the initial SETUP packet can accompany the SYN packet sent at the start of TCP's three-way handshake.

Below, we discuss these steps in more detail and consider possible optimizations.

4.2 Path Setup

Basic Setup. To setup a pinned path, the sender creates a new SNAPP header and sets the packet type to SETUP. When a router receives a SETUP packet, it first calculates the next hop r. It encodes the routing decision (as discussed

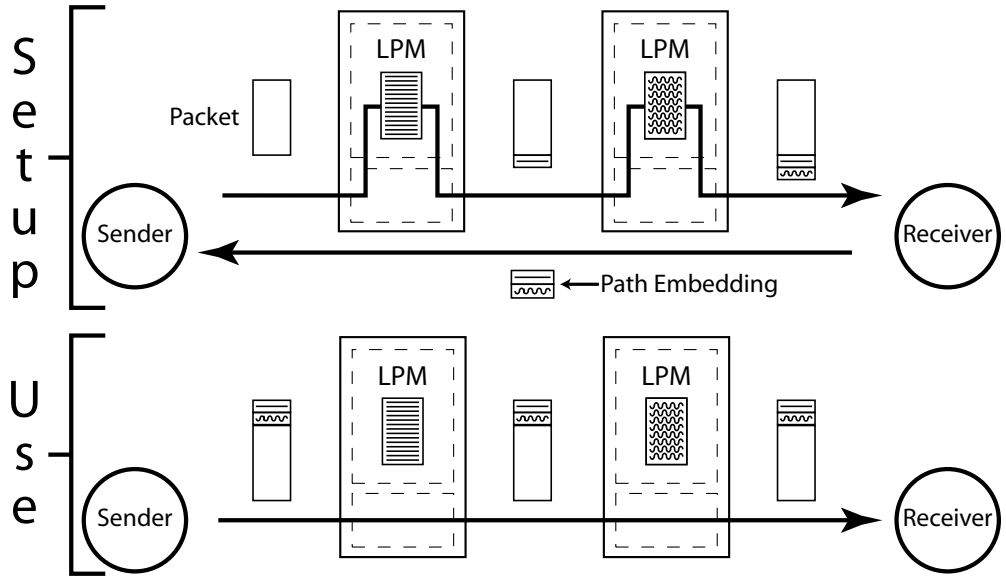

Figure 1: SNAPP Overview. ***During the setup phase, each router makes a routing calculation (e.g., Longest Prefix Match – LPM) and adds the resulting segment embedding to the packet. The receiver returns the resulting collection of segment embeddings (called a path embedding) to the sender. When a sender includes the path embedding in subsequent packets, the routers can forward the packet without performing a route table lookup.***

Algorithm 1 : Basic Router Pseudocode *These are the two main functions performed by a router. If the incoming packet p is a SETUP packet, the router invokes the* setup *routine, and if the incoming packet contains a path embedding, it invokes the* pinned *routine.*

```
1:  function setup()
2:      r ← CalculateRoute(p)
3:      r' ← EncodeRoute(r)
4:      m ← ComputeAuthenticator(r')
5:      Add segment embedding (r', m) to p
6:      Forward p based on r.
7:  function pinned()
8:      s' ← LocateMySegmentEmbedding(p)
9:      (r', m) ← s'
10:     m' ← ComputeAuthenticator(r')
11:     if m ≠ m' then
12:         Authentication failure. Drop packet.
13:     r = DecodeRoute(r')
14:     Forward p based on r.
```

below) as r' and computes an authenticator m (as discussed in §4.4). It adds its segment embedding (r', m) to the list of segment embeddings. Finally, it forwards the packet to the next hop. When the SETUP packet reaches its destination, the receiver returns the path embedding (consisting of the accumulated segment embeddings) to the sender. These steps are summarized in the setup function shown in lines 1–6 of Algorithm 1.

Encoding the Routing Decision. When a setup packet arrives, the router performs a routing calculation to determine the appropriate next hop. To create a segment embedding, we must choose an encoding for the next hop. We reject the obvious encoding of the IP address of the next hop. Such an encoding requires 32 bits, while no router has 2^{32} next-hop neighbors. Instead, in general, we encode the next hop neighbor ID based on the outbound interface chosen for the packet. For some routers, a single interface may connect via a shared medium to multiple next-hop neighbors. Such routers must maintain a small table mapping locally assigned unique identifiers to next hop neighbors.

Hiding Topological Information. As described, embedding a neighbor ID leaks topological information, since an interested party can map neighbor IDs to the corresponding routers and thus determine the path traversed by a particular packet. For many applications, this is perfectly acceptable. For applications that wish to hide this topological information, we can instead embed an encrypted version of the neighbor ID. In this version, each router maintains a secret key, known only to itself. After it selects the appropriate neighbor ID, it encrypts the ID using its secret key[1] and an encryption scheme resilient to Chosen-Plaintext Attacks (CPA). A CPA-secure scheme will produce a different ciphertext upon every invocation, even if we encrypt the same plaintext twice. Thus, an adversary will be unable to corre-

[1]For proper cryptographic hygiene, this key should be different from the key used to compute the segment authenticator, as described in §4.4.

late the encrypted neighbor ID with the neighbor to which the packet is forwarded. Technically, the encryption scheme used should be secure against Chosen-Ciphertext Attacks (CCA) as well, since the router essentially provides a decryption oracle by attempting to decrypt the contents of the neighbor ID field and route the packet appropriately. However, as discussed below, by computing the router's authenticator over the encrypted version of the neighbor ID, we can securely achieve CCA security with a CPA-secure encryption scheme, since the router will only decrypt the neighbor ID if the authenticator verifies correctly. Bellare and Namprempre demonstrate the security of this construction [5].

4.3 Forwarding

Having established a path embedding during setup, the sender can include it in later packets, allowing routers to forward the packet without making extra routing calculations.

To use a path embedding, the sender creates a new SNAPP header, sets the type field to USE, zeroes the additional fields, and appends the path embedding to the packet. When a router receives a USE packet, it locates the current segment embedding (r', m) indicated by the segment pointer field. It calculates a new authenticator m' using the same method it would use for a SETUP packet. If the two authenticators (m and m') fail to match, the router discards the packet. Otherwise, the router updates the packet's fields (§4.6), decodes r' and forwards the packet. Lines 7–14 of Algorithm 1 summarize these steps.

If the path embedding that arrives at the receiver has been modified (route or key updates, as discussed below, may modify the path embedding), the receiver must return the new path embedding to the sender. The change in the embedding alerts both the receiver and the sender that an update has taken place, unlike the current Internet in which routes can change transparently. If the path embedding arrives unmodified, the sender can use the same path embedding for multiple packets.

4.4 Preserving Path Integrity

To secure the steps described above, SNAPP must enable routers to verify that the routing decision encoded in a segment embedding is the same one calculated during path setup. It must also ensure that endhosts cannot use the segment embeddings to construct arbitrary paths through the network, as dictated by our design requirements (§3).

Authenticating the Segment Embedding. To prevent another entity from modifying the encoded routing decision, the router must also include an authenticator in the segment embedding. Since the router is the only entity that needs to authenticate its particular routing decision, the authenticator can use a secret key known only to the router. Below, we present a naïve version of an authenticator, which we refine in Equations 2 – 4.

In its simplest form, the authenticator can be a Message-Integrity Code (MIC) computed over the encoded routing decision, r', using the router's secret key, $\mathcal{K}$. Thus, a simple segment embedding s would be:

$$s = (r', MIC_{\mathcal{K}}(r')) \quad (1)$$

When a packet arrives containing the active segment embedding (r', m), the router calculates $MIC_{\mathcal{K}}(r')$ and compares it to m. If they match, the router forwards the packet to the destination encoded as r'. If they fail to match, the router drops the packet.

If the application using SNAPP employs the topologically opaque next-hop encodings discussed in §4.2, then the MIC described above should be computed over the encrypted version of the encoding. We can use this technique to convert a CPA-secure encryption scheme for the encoding into a CCA-secure encryption scheme, since the router will refuse to decrypt an encoding if the MIC does not verify properly [5].

Enforcing Path Integrity. As presented above, the authenticator for each router is independent of the other routers on the path. While this allows each router to verify the authenticity of its own routing information, it does not ensure the integrity of the entire path. Endhosts can combine different segment embeddings to create new path embeddings, violating one of our key design principles.

As a first step towards guaranteeing path integrity, we can include both the source and destination IP addresses from the packet in the computation of the router's authenticator. Thus, we can modify Equation 1 such that a segment embedding of the routing decision r' becomes:

$$s = (r', MIC_{\mathcal{K}}(r'||\text{SrcIP}||\text{DestIP})) \quad (2)$$

where $||$ denotes concatenation. This prevents an endhost from sharing a path embedding with other hosts or reusing the same path embedding to communicate with multiple destinations. However, it does not prevent endhosts from rearranging or mixing the intermediate segment embeddings from multiple path embeddings.

To fully guarantee path integrity, we make each router's authenticator depend on the previous segment embeddings. Thus, a particular router's segment embedding will only authenticate properly if the packet has traversed the same path to the router that the original setup packet followed. More concretely, suppose a setup packet arrives at a router containing a list of segment embeddings $(s_1, s_2, \ldots, s_k)$ from previous routers, $r_1, r_2, \ldots, r_k$, on the path. The current router creates a modified version of the segment embedding from Equation 2 as follows:

$$s = (r', MIC_{\mathcal{K}}(r'||\text{SrcIP}||\text{DestIP}||s_1||s_2||\ldots||s_k)) \quad (3)$$

The router can add this new segment embedding to the setup packet and forward it along. If a subsequent packet containing this embedded path arrives at the router via a path

other than $r_1, r_2, \ldots, r_k$, the set of segment embeddings in the packet will differ from $(s_1, s_2, \ldots, s_k)$, so the MIC in the packet will not match the MIC computed by the router.

4.5 Path Maintenance

Occasionally, a router may need to update the segment embedding contained within a USE packet. Security concerns dictate that each router must periodically rotate the secret key used to compute authenticators, which will necessarily change the value for those authenticators. A change in routing policy may also require a router to update its segment embedding to reflect a new routing decision.

Unfortunately, computing authenticators based on Equation 3 makes it difficult for routers to update their private keys or alter their routing decisions. If a router updates its key and then overwrites its segment embedding with the new authenticator, then the authenticators computed by subsequent routers will not match those in the packet, so it will be dropped. As an alternative, the router could leave its existing segment embedding unmodified but start a new list of segment embeddings. This could potentially double the packet overhead of SNAPP. Instead, we propose a different version of the authenticator that only requires a constant amount of space in the packet header, and we show how to use the modified authentication scheme to allow routers to perform key and route updates.

Updateable Authenticators. To facilitate maintenance of pinned paths without imposing undue overhead, we introduce a modified version of the authenticator calculation shown in Equation 3. To do so, we add an additional field $\mathcal{S}$ to the SNAPP header which will hold a copy of the segment embedding from the previous router. Thus, during setup, the sender initializes $\mathcal{S}$ to 0. When a router receives a SETUP packet, it calculates an authenticator m for its encoded routing decision r' as follows:

$$m = MIC_{\mathcal{K}}(r' || \text{SrcIP} || \text{DestIP} || \mathcal{S}) \quad (4)$$

It adds the segment embedding $s = (r', m)$ to the packet header and overwrites the contents of $\mathcal{S}$ with s.

This mechanism only requires a constant amount of additional space in the packet header, but each authenticator depends on all previous authenticators, so the property of path integrity is still preserved. As an additional side benefit, the MIC computation now occurs over a fixed amount of data, rather than the variable amount required by Equation 3. Below, we describe how to use this modified version of the authenticator to perform updates.

Key Updates. To maintain the security of the system, routers must periodically rotate the key used to create authenticators. Rather than force the sender to initialize a new SETUP packet, the router can modify its segment embedding *in situ*. After validating the current segment embedding (r', m) using the old key $\mathcal{K}$, the router calculates a new authenticator μ using Equation 4 with a new key $\mathcal{K}'$, and overwrites its old segment embedding s with the segment embedding $s' = (r', \mu)$. After sufficient time, the router can retire the old key and exclusively use the new one. However, instead of filling in $\mathcal{S}$ with s', it uses s. The next router in the path can use the contents of $\mathcal{S}$ to correctly validate its segment embedding. However, the discrepancy between $\mathcal{S}$ and the value of the previous segment embedding indicates an update has been made, so that router also updates its authenticator, using s' for the value of $\mathcal{S}$ in the new authenticator calculation. The updates continue until the packet reaches its destination. The receiver must return this new path embedding to the sender, so the sender can utilize the updated path embedding.

Route Updates. The effect a routing change has on existing path embeddings depends on an ISP's policy. An ISP may choose to apply the route update only to new connections, in which case, current path embeddings continue to function, and SNAPP provides a maximal amount of decoupling between routing and forwarding. However, the ISP may decide to apply certain types of route updates to existing connections as well. This decreases the independence of routing and forwarding, but SNAPP facilitates this type of policy as well.

When a router wishes to change the routing decision embedded in a packet, the portion of the path embedding leading to the current router remains valid, but the subsequent segment embeddings are no longer accurate. SNAPP updates the remaining entries on the fly, rather than forcing the sender to create another SETUP packet. When a packet arrives, the router makes a new route calculation and encodes it as ρ. It calculates a new authenticator μ using Equation 4 (replacing r' with ρ), and overwrites its segment embedding with $s' = (\rho, \mu)$. The router changes the packet type from USE to SETUP, telling subsequent routers to follow the setup procedure outlined in §4.2. The change in packet type also alerts the endhost that the updated path embedding represents a new route, not merely the result of a key update. The receiver must return this new path embedding to the sender, so the sender can use the new path embedding.

4.6 SNAPP Details

The SNAPP header contains a fixed-length portion with information about the packet followed by a variable length list of accumulated segment embeddings, each 32 bits wide (see Figure 2). In the fixed-length portion, a type field indicates whether the packet is of type SETUP or USE. The segment pointer indicates the currently active segment embedding. During setup, a router places its segment embedding at this location and increments the segment pointer. During forwarding, a router assumes that the active segment embedding belongs to it. If the segment embedding is authentic, the router increments the segment pointer before forwarding

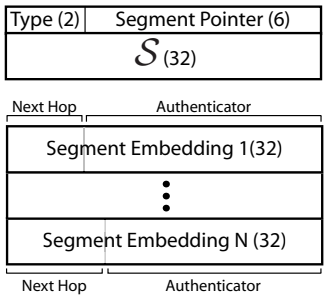

Figure 2: SNAPP Header. ***Fields contained in the header of a SNAPP packet. Each field's size in bits is shown in parentheses. The header contains one segment embedding for each router on the path. While the length of the segment embedding is fixed, the relative lengths of the next-hop encoding and the authenticator are selected by each router to maximize the size of the authenticator, given the number of next-hop neighbors.***

the packet. Finally, the header contains the field $\mathcal{S}$ that holds a copy of the segment embedding calculated by the previous router (the sender initializes it to 0). §4.5 explains how this field is used to update segment embeddings in place.

Segment embeddings consist of two parts: an encoded next-hop neighbor ID and an authenticator. Each segment embedding is 32 bits long, but the division of bits between the two parts is variable. This allows SNAPP to adapt to the fact that many routers only have a handful of neighbors, but some, such as DSL aggregating routers, may have tens of thousands of neighbors [13]. The next-hop neighbor encoding may use from 2 to 16 bits, leaving 16 to 30 bits for the authenticator. Even routers that require all 16 bits to encode the next-hop neighbor ID still have 16 bits for the authenticator, which, as we explain in §6.1, will still provide sufficient security. Since each router decodes its own segment embeddings, SNAPP does not need to waste bits to indicate the sub-field lengths: once a router is configured to use ℓ bits for the neighbor encoding, it knows that the remaining bits encode the authenticator.

To reduce space overhead, SNAPP can employ path-embedding caching nearly identical to the caching scheme used by TVA [25]. The scheme uses constant space at routers to cache the path embeddings for the largest flows, in which the overhead is most significant. The caching adds a 48-bit flow ID to the header. A sender chooses a random nonce for the flow ID, and each router includes the flow ID in the calculation of the authenticator. When a packet arrives with a valid path embedding, the router caches the flow ID along with the path embedding. The sender can omit the path embedding in subsequent packets, adding only the flow ID. This caching scheme adds complexity, since endhosts must model router cache evictions, but for some applications, the bandwidth savings may justify the additional complexity.

5. APPLICATIONS

SNAPP can serve as a building block in a wide range of applications. This section considers four applications enabled by SNAPP. We present these applications not as complete or optimal solutions, but rather as examples of SNAPP's usefulness and versatility as a basic primitive.

5.1 Sender Anonymity

With a few changes, SNAPP can provide a system for preserving sender anonymity. To do so, routers embed both forward and reverse routing decisions in packets. When a SETUP packet arrives at a destination, it also contains a reverse-path embedding. The receiver can use this embedding to reply to the initiator without knowing the initiator's network address.

For brevity, we make two simplifying assumptions before detailing the anonymity system. First, while routes between hosts may be asymmetric (e.g., because of routing policies), we assume that the network can, if necessary, provide symmetric routes. Although this assumption may not hold for some specialized networks, such as some wireless links, asymmetric satellite links, or microwave connections, most links are symmetric in the current Internet. This assumption could also hold by construction in the design of future network architectures or overlay networks. Second, we assume that routers can identify which neighbor sent them a particular packet. If an interface connects to a broadcast medium with multiple neighbors, packets from these neighbors must be distinguished by additional markings or layer 2 header information.

Given these two assumptions, we can design an anonymity system based on the SNAPP protocol. When a sender initiates a SETUP packet, its local router (or potentially a proxy) encrypts the source IP address using symmetric key encryption based on a secret key known only to the router. It then embeds the encrypted IP address in the packet, removes the original source IP address, and forwards the packet to the next-hop router.

When a SETUP packet arrives at a router, the router records the neighbor, n, from which it received the packet. It encodes the neighbor ID as n' using the opaque encodings discussed in §4.2 and computes a standard authenticator that includes n'. When the SETUP packet arrives at the recipient, it will contain a series of segment embeddings that point from the recipient to the sender. The recipient can include this reverse-path embedding in its response. Each router will locate its segment embedding and decode the routing information n' to obtain the ID n of the appropriate next-hop neighbor. The router then forwards the packet along. Eventually, the packet will reach the router local to the original sender. That router decrypts the IP address embedded in the path embedding and forwards the packet across the final hop to the sender.

The scheme described above provides efficient initiator-anonymous communication. Only the first router knows the true identity of the sender, which is reasonable in most scenarios, and even this knowledge can be weakened by using a proxy or other related techniques for anonymity [17, 21]. Each router on the path knows only the previous and next-hop routers; compromising a portion of the routers will not compromise the sender's identity. A global adversary, however, could trace a message. Resisting this adversary requires stronger cryptographic approaches such as DC-nets [9].

5.2 Sender Accountability

Many experts have opined that most current Internet threats could be addressed if the source of a packet and its contents could be reliably identified. The hope is that sender accountability would provide the foundation for eradicating worms, viruses, Denial-of-Service attacks, and other forms of Internet threats. While the effectiveness of such an approach is debatable, we design a scheme built on top of SNAPP that efficiently provides strong accountability guarantees of packet (and packet data) origin. In doing so, we illustrate the versatility of SNAPP, and in particular demonstrate its usefulness in enabling very expensive routing calculations.

Requirements. Each router can securely and statelessly identify the host responsible for *each* packet it forwards. Second, a packet recipient can prove to an ISP that a particular sender is responsible for the packet it received.

These are strong requirements that are difficult to achieve efficiently; so far, researchers have informally suggested that senders could digitally sign every packet, which would be prohibitively expensive, especially in terms of the computation required.

The first requirement implies that malicious hosts cannot masquerade as legitimate hosts, allowing routers to perform accurate filtering. Attackers must either take responsibility for their traffic or compromise legitimate hosts. The second requirement would allow an ISP to respond to customer complaints in a fair and accurate manner.

Our scheme achieves these properties while only requiring a minimal amount of computational overhead. Our key insight for designing an efficient system is to perform a relatively expensive key setup between the sender and each router on path setup, leverage SNAPP in-packet state to store the cryptographic information, and use that cryptographic state to efficiently verify subsequent packets. In essence, we amortize an expensive route establishment over subsequent packets, to achieve a viable sender accountability system.

This scheme does require additional space within the packet headers, but we believe this is a necessary tradeoff to achieve the desired accountability properties in a stateless manner. To provide accountability, we assume the presence of a Public-Key Infrastructure (PKI) which routers can access, and leave a PKI-less solution as an open problem.

Overview. This scheme expands upon the basic SNAPP protocol to establish a symmetric session key between a sender and each router along a path without requiring per-flow state at the router. During setup, the sender authenticates to the routers using an asymmetric signature. Each router uses the PKI to verify the signature, generates a symmetric session key based on the sender's ID using a pseudo-random permutation, and encrypts the symmetric key under the sender's public key. The router includes the encrypted key in the packet, and it uses SNAPP to securely embed the sender's identity in the packet.

In later packets, the sender includes the path embedding and a MIC of the packet's static contents for each router on the path. Each router uses SNAPP to verify the integrity of its segment embedding, regenerate the symmetric session key and verify the MIC on the packet. Since the segment embedding contains the sender's identity, the router can treat the packet appropriately.

Details. Senders prepare a standard SNAPP SETUP packet and include an asymmetric signature computed over the static portion of the packet's header and contents. The signature scheme is chosen such that verification is relatively inexpensive (e.g., Rabin signatures or RSA signatures with a small verification exponent). When a router receives a setup packet, it uses the sender's public signing key to verify the signature and assign the sender a local ID_S[2]. Then it generates a symmetric key $K_{SR_i} = F_{\mathcal{K}_i}(ID_S)$ that it will share with the sender, where F is pseudo-random permutation keyed by the router's secret key $\mathcal{K}_i$. The router then encrypts K_{SR_i} under the sender's public encryption key, PK_S. The asymmetric encryption scheme is chosen such that encryption is relatively inexpensive (e.g., RSA encryption). Finally, the router creates the following segment embedding (in a manner similar to the usual SNAPP embedding):

$$s = (r', ID_S, \{K_{SR_i}\}_{PK_S}, Auth(r', ID_S)) \qquad (5)$$

where r' is the usual SNAPP encoding of the routing decision, and $Auth()$ is the usual SNAPP authenticator shown in Equation 4, but with ID_S as an additional input. As usual, the receiver returns the collection of segment embeddings to the sender.

Upon receiving the collection of segment embeddings, the sender decrypts the symmetric key K_{SR_i} for each router. To send a new packet, the sender computes a MIC over the static contents of the packet p using each K_{SR_i} in turn. The packet must contain each of these MICs, along with the segment embeddings returned by the receiver. Thus, for each router on the path, the sender includes:

$$s' = (r', ID_S, Auth(r', ID_S), \text{MIC}_{K_{SR_i}}(p)) \qquad (6)$$

[2]To save space, routers could agree on a canonical identifier for each sender, so that the sender's ID would only be included in the packet header once.

When a router receives such a packet, it first verifies the authenticator. Then, it regenerates K_{SR_i} using ID_S and $\mathcal{K}_i$ and verifies the MIC on the packet. If these checks succeed, the router can attribute the packet and its contents to the sender represented by ID_S. Finally, the router forwards the packet based on the decoding of r'.

If an endhost receives a malicious packet, it can provide the packet as evidence to the ISP responsible for the router that forwarded the packet. Since the ISP controls the router, it can access the router's secret key $\mathcal{K}_i$, verify the packet's authenticator, regenerate K_{SR_i} and verify the MIC on the packet. Presumably, the ISP trusts its own routers, so it knows that sender ID_S must have originated the packet, and it can blacklist that sender in the future. While this scheme does not provide infallible legal evidence (since the ISP can lie), it does allow the party with the most power to filter network traffic to accurately identify the traffic to be filtered.

5.3 Traffic Engineering

The goal of traffic engineering (TE) is to balance network load across different paths to improve the utilization or responsiveness of the network. A challenge facing traffic engineering is that the balancing must not split a single flow across multiple paths, because TCP performs poorly in the face of packet re-ordering [26]. As a result, most TE schemes either operate offline at a granularity larger than per-flow [10], or pin flows to a particular path [12], requiring per-flow state in the router doing the splitting. Using SNAPP, a router can allow established flows to remain bound to their original path, while directing newly arriving flows to an alternate route. Of course, SNAPP does not ease the task of finding the right load balancing, but it makes it possible to do so without per-flow state. If SNAPP were configured to share authentication keys between routers, the splitting router could even define the rest of the path through an ISP, a task currently performed by using tunnels configured with a protocol such as RSVP-TE [4].

5.4 Sender-Controlled Paths

SNAPP gives senders an appropriate amount of control over the paths their traffic traverses. SNAPP's properties explicitly prevent senders from creating arbitrary paths that would violate router (or ISP) policies; instead, senders can select amongst paths provided by the network infrastructure.

Of course, we must assume the existence of a mechanism for discovering multiple network paths. One simple implementation might be for senders to periodically send SETUP packets through the network to a destination and cache the resulting path pinning for later use. Another common mechanism for finding and using multiple Internet paths is an overlay network [2, 19, 22]. Conversely, once an overlay discovers a desirable set of paths (e.g., a set whose first hops all traversed different links), it could use the path pinning to ensure that these properties persisted through routing updates.

6. ANALYSIS

In this section, we analyze the security and performance of the basic SNAPP protocol presented in §4.

6.1 Security Analysis

We first consider the effects of malicious endhosts and then examine the impact of malicious routers.

Malicious Endhosts. A malicious endhost may try to attack the availability of the SNAPP system, recombine segment embeddings to create unauthorized paths, or subvert the cryptographic primitives we employ. To attack the availability of the system, an endhost might attempt to launch a Denial-of-Service (DoS) attack on a router by forcing it to compute many authenticators. However, as discussed in the performance analysis below, a SNAPP packet requires only a single MIC computation over a fixed amount of data, which can be performed at line speed, making a DoS attack on the router infeasible. SNAPP itself is not designed to protect recipients from DoS attacks, though it does improve capability-based systems that do so, as discussed in §7. We also note that by creating a segment embedding, a router does not give up control over how a packet is forwarded; it can always reroute a packet.

The design of the segment authenticator prevents an endhost from modifying the encoded routing decision or recombining segment embeddings. Since the authenticator is always computed over the encoded routing decision, modifying the routing information will invalidate the authenticator (we discuss attacks on the authenticator itself below). Since the router always checks the authenticator before acting on the routing information, any such modification will be detected. The final version of the authenticator, shown in § 4.4 and 4.5 Equation 4, incorporates the segment embedding from the previous router. Thus, the authenticators form an authentication chain, and each new authenticator is based on all of the previous segment embeddings, preventing recombination.

Changing the value of any one segment embedding necessarily invalidates the authenticators for the subsequent routers. Suppose that a malicious endhost has two path embeddings, p and p', each consisting of a list of segment embeddings:

$$\begin{aligned} p &= (s_1, s_2, \ldots, s_k) \\ p' &= (s'_1, s'_2, \ldots, s'_n) \end{aligned}$$

If the endhost attempts to rearrange the segment embeddings within p, e.g., by swapping s_i with s_{i+1}, then there will be an authentication failure at router i. Router $i-1$ will forward the packet to router i as before, but router i will attempt to authenticate the segment embedding created by router $i+1$, which will fail with high probability as discussed below. Thus, router i will drop the packet. Now, suppose that the malicious host attempts to splice the two paths to create $\hat{p} = (s_1, s_2, \ldots, s_i, s'_j, \ldots, s'_n)$. If $s'_j \neq s_{i+1}$, then router i will forward the packet to router $i+1$, which will attempt

to authenticate the embedding s'_j. This will fail with high probability. Otherwise, when the packet arrives at router j, the router will calculate the target authenticator using s_i as the value of the previous segment embedding and compare it with the existing authenticator which was calculated using the value of s'_{j-1}. With high probability these two values will not match and the packet will be dropped. Similarly, attempting to remove segment embeddings from the beginning of path p to create $\hat{p} = (s_i, \ldots, s_n)$ will fail, since the first router will calculate the target authenticator using 0 as the previous value, whereas the existing authenticator used the value of s_{i-1}. A malicious endhost could truncate the path embedding to create $\hat{p} = (s_1, \ldots, s_i)$, but router s_i will forward the packet to router s_{i+1}. When router s_{i+1} fails to find an appropriate embedding in the packet header, it will drop the packet.

A malicious endhost could also attempt to subvert the cryptographic authenticator used by SNAPP. Since the authenticator consists of a standard cryptographic MIC, an adversary attempting to forge a correct authenticator must resort to brute force guessing. Assume that the router has over 32,000 neighbors and hence only uses 16 bits for the authenticator, even though most routers are likely to have far fewer neighbors and therefore use 25 or more bits for their authenticators. With a 16-bit authenticator, an attacker will find a correct forgery after computing $2^{16-1} = 32,768$ authenticators, in expectation. With a 25-bit authenticator, forging a single authenticator will require over 33 million attempts. Since the adversary cannot locally verify the validity of a forgery, he or she must transmit a packet for every guess, indicating that the adversary must send over 32,000 packets to find a forgery for a single 16-bit authenticator. Simultaneously forging two consecutive authenticators would require over a billion attempts. To prevent an adversary from using TTL values to probe one router at a time, we can incorporate the value of the TTL field in the authenticator's MIC, forcing the adversary to successfully forge authenticators for the entire path. The amount of work required for a successful forgery makes this attack prohibitively expensive, particularly since periodic router key changes will invalidate the work done.

Malicious Routers. Much of the above analysis also applies to malicious routers. A router might also misroute a packet or alter the packet's SNAPP-related data. If a router reroutes a packet, it will arrive at a legitimate router, but the segment embedding pointed at by the segment pointer will not belong to that new router. Since the legitimate router has a different key from the intended next-hop router, the authenticator will fail to validate with high probability, and the probability of two successive authenticators validating successfully is negligible. When the authenticator fails to validate, the router will drop the packet. While this is certainly not ideal from the sender's perspective, it is also no worse than if the malicious router decided to drop the packet itself.

A similar analysis applies to modifications of SNAPP data. A malicious router can cause a packet to be dropped by an earlier router on the path, but the malicious router could just as easily drop the packet itself. Modifying an earlier router's embedding during setup will cause later packets to be dropped when they reach the earlier router. Similar modifications to non-setup packets will have no effect. Modifying a later segment embedding will cause an authentication failure since the authenticator will fail to validate, resulting in a dropped packet. As discussed in §5.4, SNAPP supports sender-selected paths; such a mechanism would allow the sender to avoid a path that drops too many packets.

Finally, since SNAPP does not require any trust relationships between routers, subverting one router does not aid the adversary in subverting (or otherwise affecting) other legitimate routers.

6.2 Performance Analysis

SNAPP can operate at line-speed and requires a reasonable amount of space in packet headers, an amount that can be further reduced to only six bytes through the use of caching.

When a setup packet arrives, the router must perform the usual routing calculation. It must also compute the MIC shown in Equation 4 over 112 bits of input. When a packet arrives containing a path embedding, the router must perform the same MIC computation and compare the result to the packet contents, but it no longer needs to perform a routing calculation. Thus, the amount of time and state required to forward a packet is no longer dependent on the size of the routing table, but instead depends only on the time required to compute a single MIC.

A single MIC can be easily implemented in hardware to provide line-speed performance. A typical block-cipher-based MIC such as CBC-MIC requires two serial invocations of a block cipher such as AES. ASIC cores for AES can operate at data rates ranging from 30 Mbps to 25 Gbps and require between 6,000 and 30,000 gates [11]. We can further optimize the MIC computation by using PMAC, an alternative mode of operation for block-cipher based MICs that is fully parallelizable and hence highly efficient to implement in hardware [6]. Thus, the MIC computation will not create a bottleneck on the router's throughput.

Examining space overhead, a standard SNAPP header requires five bytes plus four additional bytes for each router traversed. As discussed in §4.6, after the initial round, routers can cache the path embedding, so that the sender need only include a six-byte flow ID in most of the packets sent, but the routers still use a constant amount of state.

7. DISCUSSION

This section examines some of the additional useful properties that SNAPP provides. We also consider an ISP's control over routing decisions once SNAPP is deployed. Finally, we discuss the effectiveness of SNAPP in incremental deployments.

Availability. By allowing forwarding to be decoupled from routing, SNAPP enhances network availability. As discussed in Section 4.5, ISP policy ultimately determines the extent of the decoupling. With a policy of maximal decoupling, an attack on routing (or a misconfiguration [7, 27]) will not interrupt existing flows, since the routers will continue to forward their packets based on the earlier routing decision encoded in the packets. As long as a sender possesses a valid path embedding leading to the recipient, its packets can continue uninterrupted by problems on the routing plane. Senders can cache recently used path embeddings so that setup is performed infrequently.

Network Measurement. SNAPP can also facilitate network measurements. A measurement tool could pin a path in place and eliminate path variability as a source of measurement noise and/or error. Any change in the path (as the result of a routing policy update) will be reflected in the modified value of the path embedding, so that the change can be properly factored into the measurements. The change in the path embedding would also allow a measurement tool immediately know when path changes occur, which may also provide useful insight into network characteristics.

Capability Systems. SNAPP addresses an important limitation of capability systems: their fragility under path changes. Capability systems allow a receiver to provide legitimate senders with a capability token that authorizes them to send privileged network traffic. Each capability is typically tied to a particular path between the sender and receiver. Any deviation from the path invalidates the capability. Using SNAPP, a sender can pin the path in place, preserving the capability even in the face of routing problems or transient routing changes.

Since most capability systems require routers to compute and check an authenticator for information embedded in a packet, the router functions required for capabilities and for SNAPP overlap and could be easily combined.

ISP Route Control. By implementing SNAPP, ISPs do not give up control over how routes are determined. Indeed, the ISP provides the various routes in the first place, and one of SNAPP's explicit design goals (§3) prohibits endhosts from constructing arbitrary paths. Furthermore, even after a path has been pinned in place, a router may still decide to reroute the flow's traffic, though this decision will increase the dependence between forwarding and routing. As explained in §4.5, our design includes the flexibility to update the path embedding dynamically in place in order to accommodate such changes.

Incremental Deployment. While we primarily envision SNAPP as a building block for new architectures, we also believe it has value in incremental deployments. The effectiveness of a path embedding depends on the number of SNAPP routers along the path, as well as the stability of routes through the non-SNAPP routers. The non-SNAPP routers do not embed their routing decision in setup packets, and hence they may make different routing decisions for subsequent packets. Thus, path embeddings for paths with a small fraction of SNAPP routers that make frequent route changes may prove unstable. Nonetheless, many path embeddings will be sufficiently stable over the lifetime of a connection, and SNAPP routers, particularly those close at the edge of the network, can still benefit from SNAPP's decoupling of forwarding from routing, as well as the amortization of expensive initial route lookups over subsequent packets.

8. CONCLUSION

In considering architectural primitives for designing a network architecture, whether for an overlay network or a next-generation network core, we find that SNAPP represents a versatile building block for achieving a number of useful properties. SNAPP provides sufficient flexibility to: 1) decouple forwarding from routing to enhance the availability of paths in the face of routing disturbances, 2) provide route-selection control to the sender (to request multiple routes and select among them), 3) enable applications with expensive route lookups, 4) provide capability-based systems with stable paths despite routing changes, 5) enable load balancing at the sender, 6) provide sender anonymity at the network layer, and 7) provide sender accountability.

SNAPP also provides additional flexibility for implementing routers and other forwarding devices; for example we can envision a system where high-speed switches perform the packet forwarding, and separate servers are used to aid in path setup. This may lead to an approach for optical networks, where switching may be feasible in the all-optical domain, whereas the more complex routing decisions occur in traditional hardware.

9. REFERENCES

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proc. ACM SOSP*, 1999.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. ACM SOSP*, 2001.

[3] K. Argyraki and D. R. Cheriton. Loose source routing as a mechanism for traffic policies. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Sept. 2004.

[4] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. *RSVP-TE: Extensions to RSVP for*

LSP Tunnels. Internet Engineering Task Force, Dec. 2001. RFC 3209.

[5] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Proceedings of AsiaCrypt*, 2000.

[6] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *Advances in Cryptology - EUROCRYPT*. Springer-Verlag, 2002.

[7] V. J. Bono. 7007 explanation and apology. `http://www.merit.edu/mail.archives/nanog/1997-04/msg00444.html`, Apr. 1997.

[8] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. *Resource ReSerVation Protocol (RSVP)*. IETF, 1997. RFC 2205.

[9] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Jrnl of Cryptology*, I(1), 1998.

[10] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE J-SAC*, 20(4):756–767, May 2002.

[11] Helion Technology Limited. High performance AES (Rijndael) cores for ASIC. Cambridge, England. Available at `http://www.heliontech.com/`. March 2007.

[12] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *Proc. ACM SIGCOMM*, Philadelphia, PA, Aug. 2005.

[13] L. Li, D. Alderson, W. Willinger, J. Doyle, R. Tanaka, and S. Low. A first principles approach to understanding the Internet's router technology. In *Proceedings of ACM SIGCOMM*, 2004.

[14] NSF workshop report. Overcoming barriers to disruptive innovation in networking, Jan. 2005.

[15] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *Proceedings of the ACM SIGCOMM*, Aug. 2007.

[16] B. Raghavan and A. C. Snoeren. A system for authenticated policy-compliant routing. In *Proc. of ACM SIGCOMM*, 2004.

[17] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM TISSEC*, 1:66–92, Nov. 1998.

[18] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. RFC 3031, Jan. 2001.

[19] I. Stoica, D. Adkins, S. Zhaung, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM*, Aug. 2002.

[20] I. Stoica and H. Zhang. Lira: An approach for service differentiation in the internet. In *Proceedings of NOSSDAV*, June 1998.

[21] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, 1997.

[22] J. Touch and S. Hotz. The X-Bone. In *Proc. 3rd Global Internet Mini-Conference in conjunction with IEEE Globecom*, 1998.

[23] A. Yaar, A. Perrig, and D. Song. SIFF: An endhost capability mechanism to mitigate DDoS flooding attacks. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2004.

[24] X. Yang. *NIRA: A New Internet Routing Architecture*. PhD thesis, MIT, Sept. 2004.

[25] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *Proceedings of ACM SIGCOMM*, Aug. 2005.

[26] M. Zhang, B. Karp, S. Floyd, and L. Peterson. A reordering-robust TCP with DSACK. In *IEEE ICNP*, 2003.

[27] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. An analysis of BGP Multiple Origin AS (MOAS) conflicts. In *Proc. of Internet Measurement Workshop*, 2001.